©2002 Microsoft Corporation

.NET Security Advanced Topics
This module is beyond the scope of the normal 2-day seminar. It is included in the interest of completeness
2

NET Security Advanced Topics

This module is beyond the scope of the normal 2-day seminar. It is included in the interest of completeness.

 ©2002 Microsoft Corporation

Advanced Topics Agenda

The majority of this module is about Evidence-based Security with its policy model for granting permissions, extensibility and customization, along with Code Access Security and the authorization enforcement mechanism for partially-trusted code. There are additional security measures afforded by .NET, however, such as Obfuscation and Type Safety that will be discussed.

The **Cryptographic namespace** not only is used by .NET to provide Role-based Security, but is available to .NET developers for use in their applications as well as for discovering and providing security settings in conjunction with operating code.

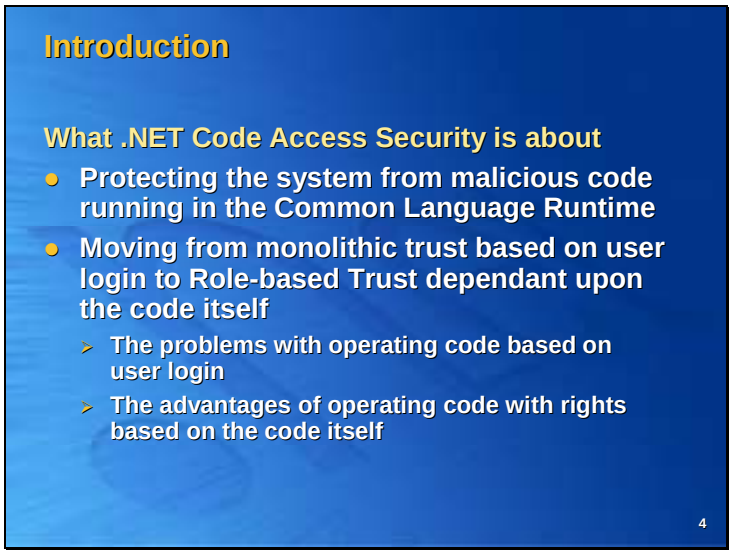
Obfuscation protects the developer's intellectual property.

Enforcement of security is a primary consideration in system reliability. Type Safety contributes toward system reliability.

These topics are discussed further at the end of the presentation.

Security scenarios covered by Evidence- and Code–based Security:

ISP Hosting - wants to ensure that applications can view or modify data belonging to others. .NET can do that.


Introduction
What .NET Code Access Security is about
Protecting the system from malicious code running in the Common Language Runtime
Moving from monolithic trust based on user login to Role-based Trust dependant upon the code itself
The problems with operating code based on user login
The advantages of operating code with rights based on the code itself
4

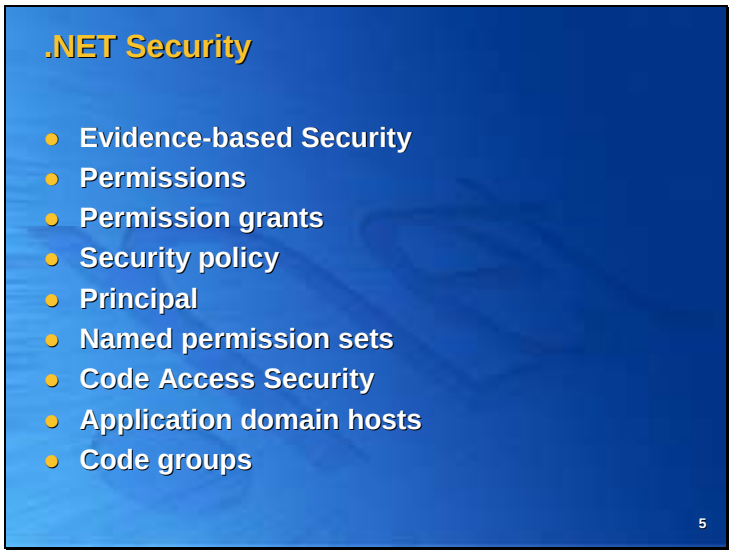
Introduction

Security scenarios covered by Evidence- and Code–based Security continued …

Possible security concerns :

- An application can access a resource and corrupt it. The list of vulnerable resources includes:
  - Message Queue
  - Event Log
  - Performance Counter
  - Active Directory
- An application can stop an important system service.
- An application can change priorities of the system processes.
- An application can use FileSystemWatcher to gather and disclose sensitive information about the file system
- Code comes from multiple sources. It can be imbedded in email, contained in documents or downloaded from the Internet, but be careful because it can be intentionally or accidentally malicious in that it can be designed to act as a Trojan horse capturing and forwarding passwords, replicating itself or performing other mischief. It also can cause damage through “bugs.”
- Traditional security afforded by operating systems takes into account only the identity of the user of the code. In other words, if the user is trusted so is the code. .NET Security provides different codes and levels of trust, not solely according to the rights of the user operating the code, but also the code’s author and system policies.

 ©2002 Microsoft Corporation



.NET Security

Security addresses the smallest details and it is vital if one is to understand the terminology and avoid semantics. Additionally, one must understand the basic functionality and the intended mechanism for using that functionality. The terminology and concepts necessary for an understanding of .NET Security are listed as Core Concepts.

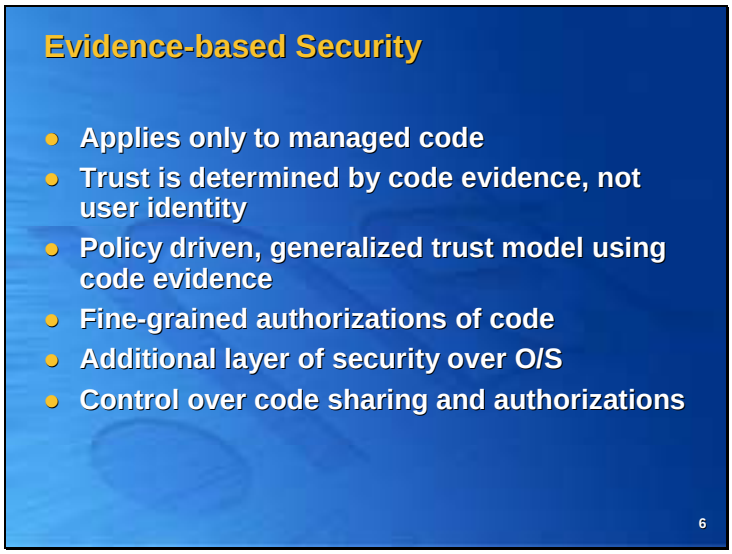
Permissions

The CLR allows code to perform only those operations it has permission to perform. To enforce restrictions on managed code, the CLR uses permissions.

The primary uses of permissions are as follows:

- code can *request* the permission it needs to access resources or perform operations
- the CLR can *grant* permission to code based on characteristics of the code's identity, on what permissions were requested and on how much the code is trusted
- code can *demand* its callers have permission

For more information about how CLR decides which permissions to grant, see Security Policy and Permission Grants.


Evidence-based Security
Applies only to managed code
Trust is determined by code evidence, not user identity
Policy driven, generalized trust model using code evidence
Fine-grained authorizations of code
Additional layer of security over O/S
Control over code sharing and authorizations
6

Evidence-based Security

Evidence is information that serves as input to the Common Language Runtime's mechanism for making decisions based on security policy, and it indicates that code has a particular characteristic. Common forms of evidence include digital signatures and the location from which code originates, but evidence also can be custom designed to represent other information that is meaningful to the application.

Extensible Evidence-based Security model

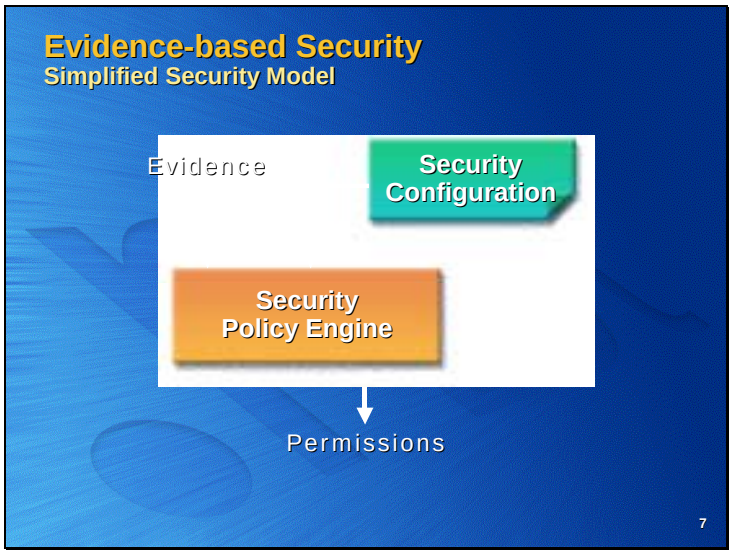
- Evidence is combined with policy to compute permissions granted to an assembly
- Evidence comes from hosts and assemblies
- Policy is merged from enterprise, machine, user and application
- Permission instances represent fine-grained authorizations

Signed and encrypted XML

- XMLDSig: W3C/IETF standard in process
- XML encryption proposed W3C WG

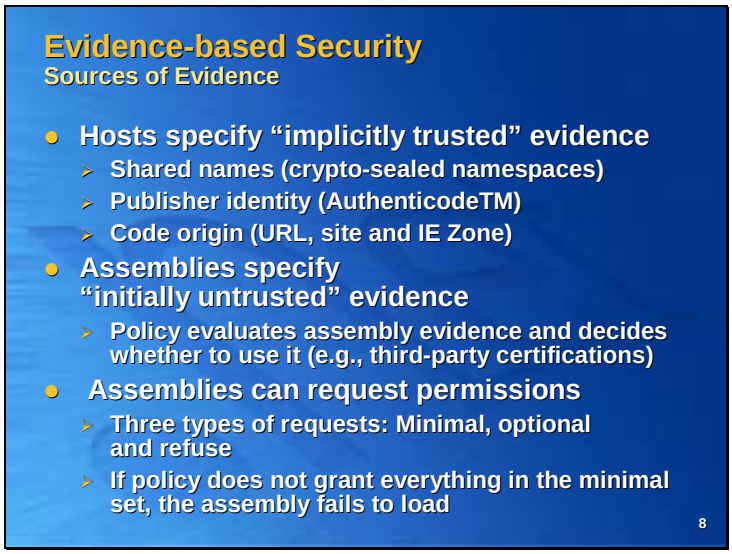
Policy is configurable by administrators.

 ©2002 Microsoft Corporation



Evidence-based Security

Simplified Security Model


Evidence-based Security
Sources of Evidence
Hosts specify "implicitly trusted" evidence
Shared names (crypto-sealed namespaces)
Publisher identity (AuthenticodeTM)
Code origin (URL, site and IE Zone)
Assemblies specify "initially untrusted" evidence
Policy evaluates assembly evidence and decides whether to use it (e.g., third-party certifications)
Assemblies can request permissions
Three types of requests: Minimal, optional and refuse
If policy does not grant everything in the minimal set, the assembly fails to load
8

Evidence-based Security

Sources of Evidence

Trusted application domain hosts can present evidence about an application domain that enables the CLR to decide what permissions to grant the application domain. This information enables the CLR to evaluate machine and user policy and return the set of permissions to grant to the application domain. If the host does not have permission to provide evidence, the application domain will get the permissions that have been granted to the host.

The CLR gets evidence about assemblies from trusted application domain hosts or from the loader. Some evidence, such as from where the code originates, needs to come from the application domain host because only the host knows. Other evidence, such as an assembly's digital signature, is inherent in the code itself, so it can come from the loader or a trusted host. Typically, when code is loaded, each assembly's digital signature is validated by the runtime. If the digital signature is valid, the host passes the signature information as evidence to the runtime's policy mechanism. In addition, an assembly or a host can provide custom evidence as a resource that is part of the assembly. Administrators and developers can define these types of evidence and extend security policy to recognize and use it.

The following types of evidence can be presented by the host: **Evidence Description**

PublisherSoftware publisher signature, i.e., the AuthentiCode™ signer of the code.

Site - Site of origin, such as www.microsoft.com

StrongName - Cryptographically strongname of assembly

URL - URL of origin

Zone - Zone of origin, such as Internet Zone

Hash - Cryptographic hash: MD5 or SHA1

Application Directory - The directory in which the application is installed

CustomApplication or system-defined evidence

 ©2002 Microsoft Corporation

The CLR's policy mechanism combines evidence from the host and assembly and uses that evidence to determine the code groups of which the code is a member. Ultimately, code group membership together with requested permissions determine which permissions the code is granted.

Assemblies can request permissions. Three types of requests are minimal, optional and refuse.

If policy does not grant everything in the minimal set, the assembly fails to load

Assembly is granted: (MaxAllowed (Minimal Optional)) – Refused

Assemblies can carry evidence

Assembly evidence is initially untrusted

Policy evaluates assembly evidence and decides whether to use it

Example: Third-party certifications

Permissions

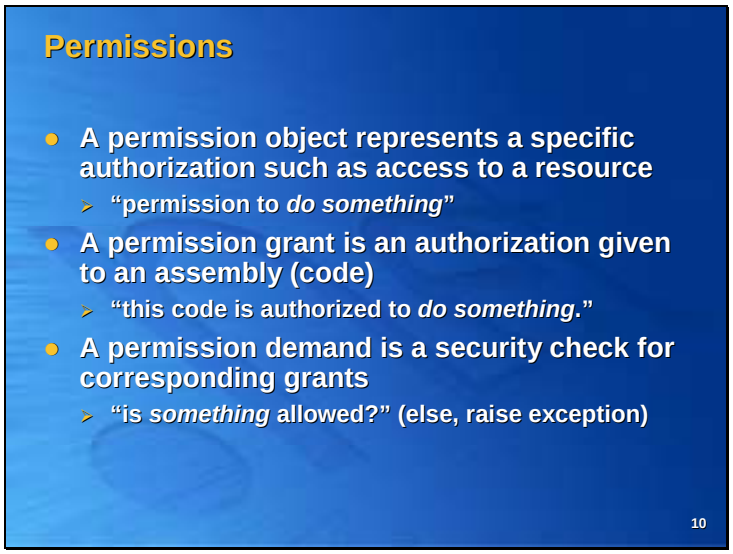
There are three kinds of permissions, each with a specific purpose:

- **Code Access Permissions** represent the right of code to access a protected resource or perform a protected operation.
  - Permission class name - Right represented
  - DnsPermission - Access to DNS (Domain Name System)
  - EnvironmentPermission - Read or write environment variables
  - FileDialogPermissionAccess - Files selected by the user in an open dialog box
  - FileIOPermission - Read, append or write files or directories
  - IsolatedStoragePermission - Access isolated storage, which is storage associated with a specific user and with some aspect of the code's identity such as its Web site, publisher or signature
  - ReflectionPermission - Discover information about a type at run time
  - RegistryPermission - Read, write, create or delete registry keys and values
  - SecurityPermission - Execute, assert permissions, call into unmanaged code and skip verification as well as other rights
  - SocketPermission - Make or accept connections on a transport address
  - UIPermission - Access user interface functionality
  - WebPermission - Make or accept connections on a Web address
- **Identity permissions** indicate that code has credentials that support a particular kind of identity.
  - Class Name - Identity represented
  - PublisherIdentityPermission - The software publisher's digital signature
  - SiteIdentityPermission - The Web site where the code originated

 ©2002 Microsoft Corporation

  - StrongNameIdentityPermission - The strong (shared) name of the assembly. For more information, see What is a shared name? ZoneIdentityPermission - The zone where the code originated. For more information, see System.Security.SecurityZone
  - URLIdentityPermission - The URL where the code originated including the protocol prefix — http, https, ftp, and so on

- **Role-based Security Permissions** provide a mechanism for discovering whether a user, or the agent acting on the user's behalf, has a particular identity or is a member of a specified role.
  - PrincipalPermission is the only Role-based Security permission.
  - The CLR provides built-in permission classes in the System.Security.Permissions namespace and also supplies support for designing and implementing custom permission classes.


Permissions
A permission object represents a specific authorization such as access to a resource
"permission to do something"
A permission grant is an authorization given to an assembly (code)
"this code is authorized to do something."
A permission demand is a security check for corresponding grants
"is something allowed?" (else, raise exception)
10

Permissions

Framework permissions for resources

- DataAccess - EventLog - DirectoryServices
- Environment - Registry - Reflection
- FileIO - FileDialog - IsolatedStorage
- UI - Printing - MessageQueue
- Socket - DNS - Web
- **Security:** execution, skip verification, assert, unmanaged code, control thread, control evidence, control policy and control principal

User identity

- Principal permission

Identity permissions indicate code identity.

- Publisher, StrongName, Site, URL and Zone

 ©2002 Microsoft Corporation

Permissions

The runtime attempts to discover what permissions policy allows code to have by traversing the code group hierarchies for the relevant policy levels. For application domains, the relevant policy levels are the machine and user policy levels; for assemblies, the machine, user and application domain policy levels are relevant. The runtime computes an allowed permission set for each relevant policy level and then combines those permission sets to obtain the allowed permission set for the assembly or application domain.

For each relevant policy level, the runtime discovers which code groups the code is a member of based on the identity information provided by the evidence. If the code is a member of a code group, that group is said to be a "match." The permission sets associated with the matching code groups determine which permissions the security policy level allows the code to have.

The search for a "match" in the code group hierarchy begins at the top of the hierarchy in the "all code" group. The levels in the code group hierarchy are searched, looking for matches in child groups as long as a match is found in a parent group. For code groups without attributes the process works like this: if a parent code group is a match, all of the matching group's children are examined to see if any of the child groups match and so on. When all matches in the hierarchy have been identified, the permissions associated with each matching code group are unioned resulting in the set of permissions allowed by that level of policy.

The following figure illustrates a code group hierarchy where Microsoft Money is a member of four code groups: All Code, Microsoft (the Publisher), Local Internet (the Zone) and Microsoft Money (the name). The allowed permission set for a given policy level (machine, user or application domain) is the union of the named permission sets associated with each of these code groups, which in this case is P1 P2 P4 P7.

Permission Grants

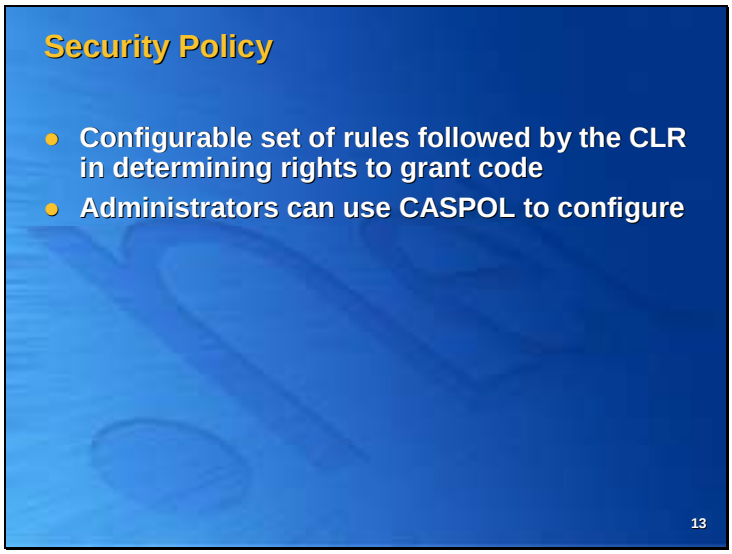
Note: if exactly one matching code group in the policy level has the *Exclusive* attribute, only the permissions of the exclusive code group apply. In this case, the allowed permissions set for the policy level is the set of permissions belonging to the exclusive code group.

Once the allowed permission sets for each policy level have been determined, the code hierarchy search is repeated for each relevant policy level and the allowed permission sets for each policy level are intersected. The resulting allowed permission set contains only the permissions that the allowed permission sets for all policy levels have in common.

If a code group has the *LevelFirst* attribute, the allowed permission set is computed differently: no policy levels below the one containing the exclusive code group are considered when checking for matches and granting permissions. For example, if a code group in the hierarchy for the machine policy has the *LevelFirst* attribute and some code matches the membership condition of this code group, neither the user level policy nor the application domain policy would be applied to that code.

For application domains, the granted permission set is the allowed permission set. For assemblies, the other factors described in this topic are considered at assembly load time when determining the grant set.

 ©2002 Microsoft Corporation


Security Policy
Configurable set of rules followed by the CLR in determining rights to grant code
Administrators can use CASPOL to configure
13

Security Policy

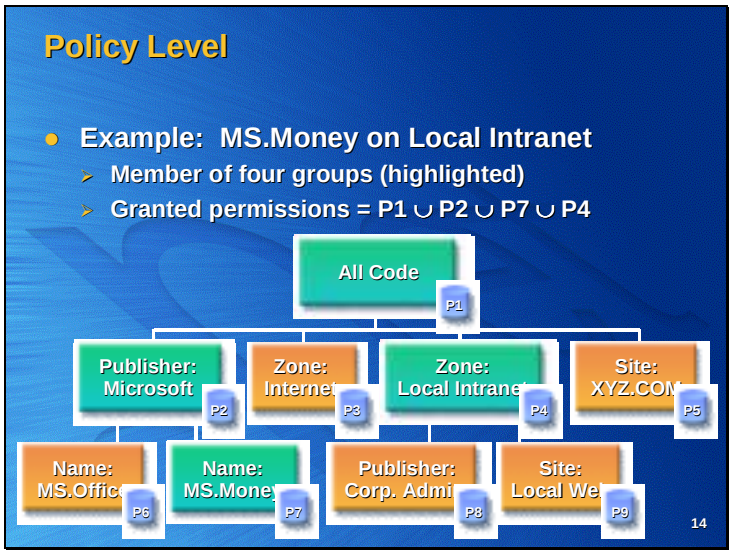
As described in the Security Policy Model topic, there are three levels on which security policy is specified: machine policy, user policy and application domain policy and each policy level has its own settings. The runtime intersects the settings for these three levels to determine the permissions that an assembly is allowed. The allowed permission set for application domains is determined by intersecting only the settings for the user and machine policy levels.

By default, both the user policy and the application domain policy specify that the set of permissions is allowed. Machine policy is set by default to the values shown in the table below. Because the policy levels are intersected when determining the allowed permission set, the machine policy settings determine default security policy.

The following table shows how the machine policy is configured by default. It indicates that, for example, code from the Internet zone is allowed by default to have the permissions in the Internet named permission set, which include file dialog (read-only), isolated storage, security permission to execute and UI permission to create safe subwindows and use the clipboard.

Security policy is the configurable set of rules that the Common Language Runtime follows when it decides what code can do. Administrators set security policy, and the runtime enforces it. The runtime ensures that code can access only the resources and call only the code that security policy allows it to access.

Whenever an attempt is made to load an assembly, the runtime uses security policy to determine which permissions to grant to the assembly. After examining information called *evidence* that describes the assembly, the runtime uses security policy to decide how much the code is trusted and, therefore, what permissions to grant to that assembly. Evidence includes, but is not limited to, the code's publisher, its site and its zone. Security policy also determines which permissions to grant to application domains.



Policy Level

Policy is the process of determining the permissions to grant code.

- Permissions granted to code, not user.
- Grants are on a per-assembly basis.

Multiple levels of policy

- Machine-wide and user-specific by default
- Further policy restrictions allowed on a per-application domain basis

Each policy level is a collection of code groups.

- Code has identity in the runtime just like users have identity in Windows NT®.
- Permissions are associated with each code group.

Evidence determines group membership.

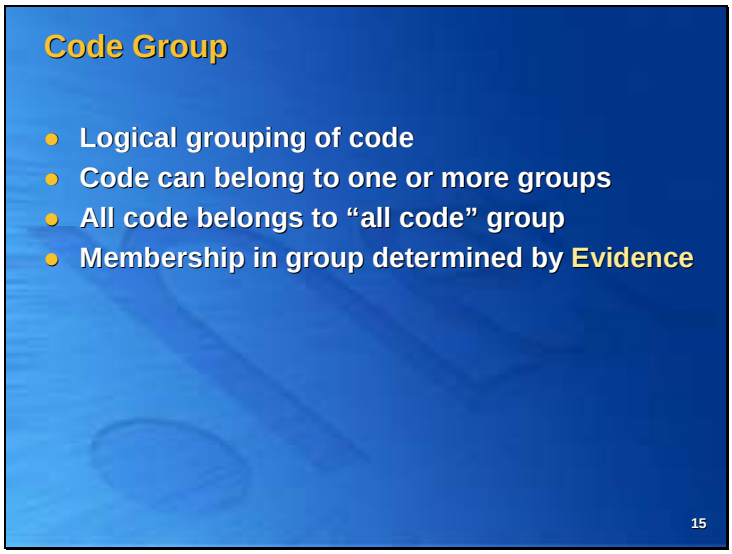
- In the group, get granted the related permissions.

Permissions granted to assembly and not user.

Evidence determines *code group*

- Code groups are hierarchical.
- Parent group merges child group permissions with its own.
- Parent defines merge logic.

 ©2002 Microsoft Corporation


Code Group
Logical grouping of code
Code can belong to one or more groups
All code belongs to “all code” group
Membership in group determined by Evidence
15

Code Group

A **code group** is a logical grouping of code in which a specified condition must be met for code to be a member of the group. Any code that meets the membership condition is included in the group. Each **code group** is also associated with a named permission set, and code groups can have attributes, which affect how the code group defines policy. Administrators configure security policy by managing code groups.

The runtime uses **evidence** describing the code to determine whether a group's **membership** condition has been met. For example, if the membership condition of the code group is "Code that is from the www.microsoft.com Web site," the runtime examines the evidence to determine whether the code does, in fact, originate from www.microsoft.com. The runtime decides which permissions can be granted to code by determining which code groups the code belongs to and then combining the permission sets associated with the groups of which code is a member.

Each type of policy (machine, user and application domain) is represented by a hierarchy of code groups. The root of each hierarchy is the group containing all code. The "all code" group has child nodes, and those child nodes have children, and so on. If code is a member of the parent code group then the code might be a member of one or more of that group's child code groups. If code is not a member of the parent code group, it cannot be a member of any of the code groups that are descended from that parent.

Code groups can be defined to have the following kinds of membership conditions:

Software publisher - Based on public key of a valid Authenticode™ signature

Zone of origin of the code

Strong name - Based on a cryptographically strong signature.

Web site of origin of the code, e.g. www.microsoft.com or *.microsoft.com

URL of origin of the code including the final wildcard, i.e., "http://site/app/*"

Cryptographic Hash - Based on an MD5 or SHA1 cryptographic hash

Application Directory - Based on the directory in which the application is installed

Custom - Defined by an application or system

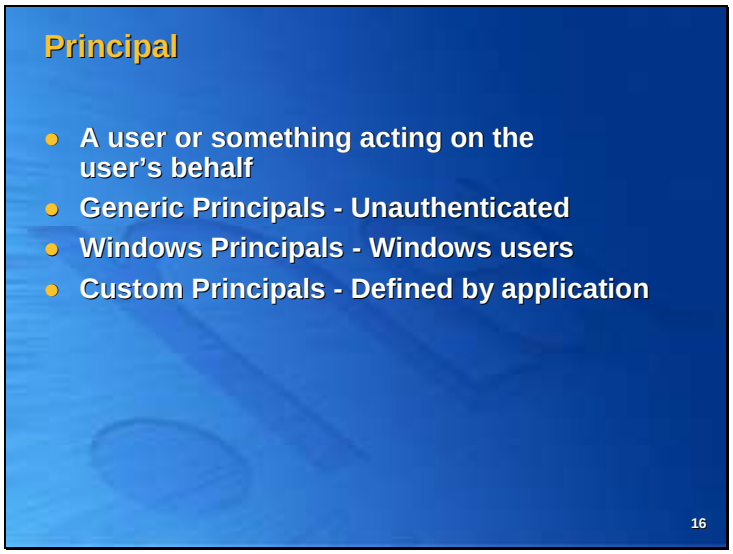
Code groups can have the following attributes, which affect how allowed permission sets are determined:

Exclusive: If code is a member of a code group that is marked Exclusive, the allowed permission set for the policy level will be the permission set associated with that code group. When all policy levels are taken into account, the code will never be allowed more permissions than those associated with the Exclusive code group. Within a given policy level, code can be a member of no more than one code group that has the Exclusive attribute.

LevelFinal: No policy levels below the one containing this code group are considered when checking code group membership and granting permissions; machine policy is the highest level of policy, followed by user policy, then application domain policy. For example, if the LevelFinal attribute is set for a code group in machine policy and some code matches the membership condition of this code group, neither the user level policy nor the application domain policy would be applied to that code.

A code group can be marked with both the Exclusive and LevelFinal attributes.

 ©2002 Microsoft Corporation


Principal
A user or something acting on the user's behalf
Generic Principals - Unauthenticated
Windows Principals - Windows users
Custom Principals - Defined by application
16

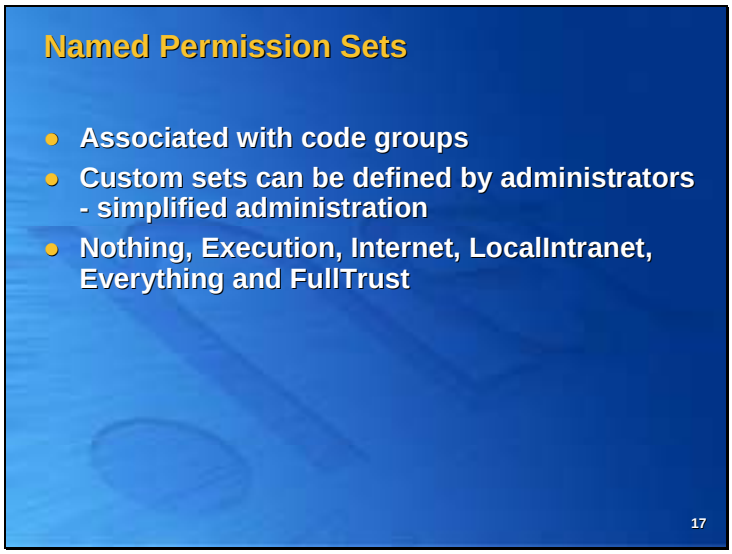
Principal

A **principal** is a user or agent that acts on the user's behalf. .NET Framework Role-based Security offers support for three kinds of principals:

Generic principals represent unauthenticated users and the roles available to them.

Windows principals represent Windows users and their roles (or their Windows NT/Windows 2000 groups). A Windows principal can impersonate another user, which means the principal can access a resource on a user's behalf while presenting the identity that belongs to that user.

Custom principals can be defined by an application in any way necessary. They can extend the basic notion of the principal's identity and roles, however, the application must provide an authentication module and types that implement the principal.


Named Permission Sets
Associated with code groups
Custom sets can be defined by administrators - simplified administration
Nothing, Execution, Internet, LocalIntranet, Everything and FullTrust
17

Named Permission Sets

Administrators can associate code groups with a *named permission set*, which consists of at least one permission and a name and description for the permission set. This association allows administrators to establish or modify the policy that determines which permissions the runtime allows code in those groups to have. More than one code group can be associated with the same named permission set since the runtime provides several built-in named permission sets, whose descriptions are as follows:

Nothing: No permissions (cannot operate)

Execution: Only permission to operate, but no permissions that allow use of protected resources.

Internet: The default policy permission set suitable for content from an unknown origin.

LocalIntranet: The default policy permission set used within an enterprise.

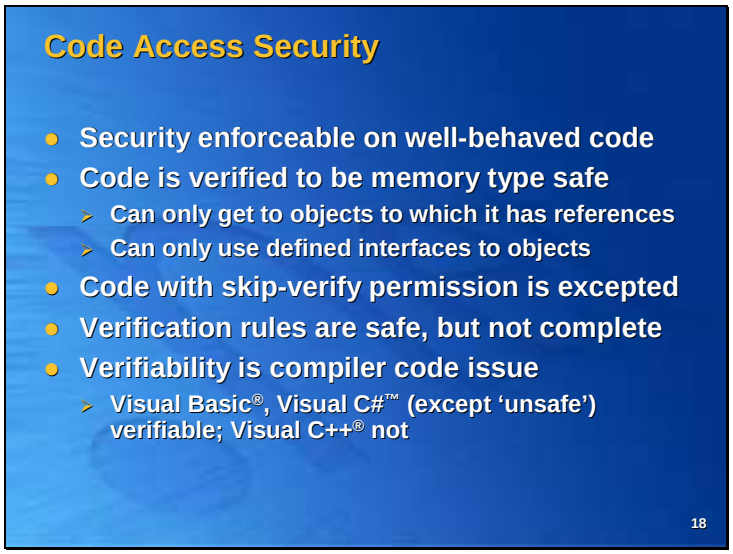
Everything: All standard (i.e., built-in) permissions except permission to skip verification.

FullTrust: Full access to resources protected by unrestricted permissions.

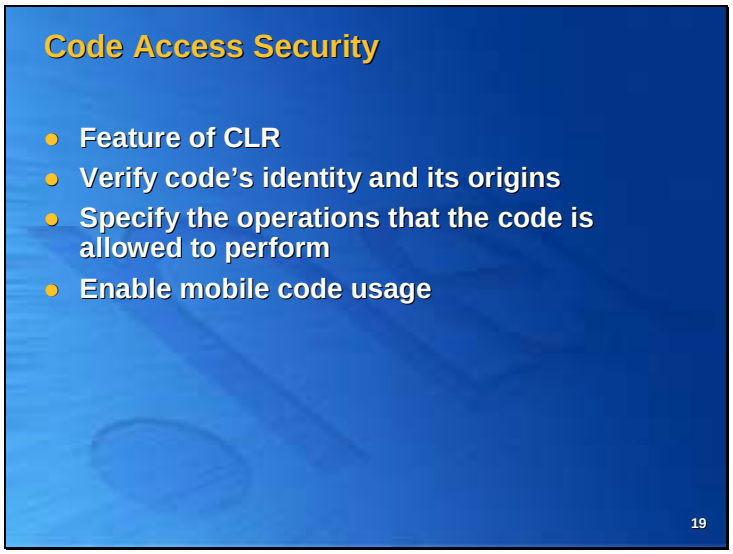
Of these built-in named permission sets, only Internet, LocalIntranet and Everything can be modified. Modifying an existing named permission set affects the permissions the runtime can grant to all code that is a member of an associated code group.

Custom named permission sets can be defined by administrators as long as the names do not conflict with the named permission sets. Named permission sets cannot contain identity permissions.

 ©2002 Microsoft Corporation


Code Access Security
Security enforceable on well-behaved code
Code is verified to be memory type safe
Can only get to objects to which it has references
Can only use defined interfaces to objects
Code with skip-verify permission is excepted
Verification rules are safe, but not complete
Verifiability is compiler code issue
Visual Basic®, Visual C#™ (except 'unsafe') verifiable; Visual C++® not
18

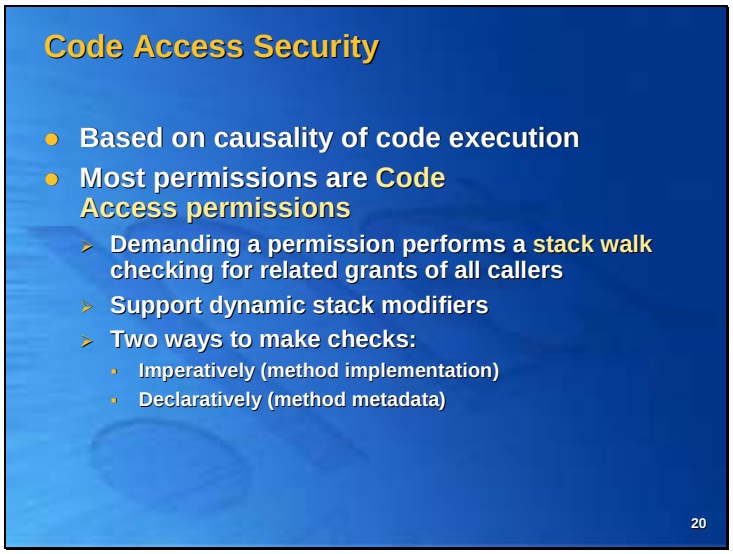
Code Access Security


Code Access Security
Feature of CLR
Verify code's identity and its origins
Specify the operations that the code is allowed to perform
Enable mobile code usage
19

Code Access Security

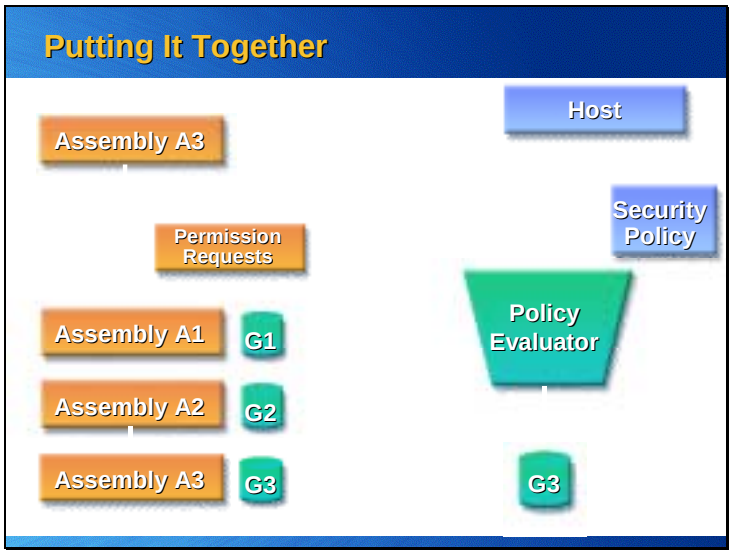
In the traditional server, all applications were trusted, yet in a hosted environment, applications from diverse sources may be managed in the same process space by the CLR. Additionally, complex applications have varied security needs. Under CLR management, applications are granted different levels of access and the CLR won't execute unverifiable code. Mobile code is software that's built using a .NET language. It is managed code that operates in the CLR and can be operated in many environments – a notebook computer, a PDA, a cell phone, even a refrigerator.

 ©2002 Microsoft Corporation


Code Access Security
Based on causality of code execution
Most permissions are Code Access permissions
Demanding a permission performs a stack walk checking for related grants of all callers
Support dynamic stack modifiers
Two ways to make checks:
Imperatively (method implementation)
Declaratively (method metadata)
20

Code Access Security

Most permissions are Code Access permissions.



Putting It Together

Evidence is the input to policy.

Example: Info about a code assembly

- Shared names
- Publisher identity
- Location of origin (URL, zone and site)

Evidence is completely extensible.

- Any object can be a piece of evidence.
- Only impacts grants if a code group membership condition cares about it

Hosts can influence policy.

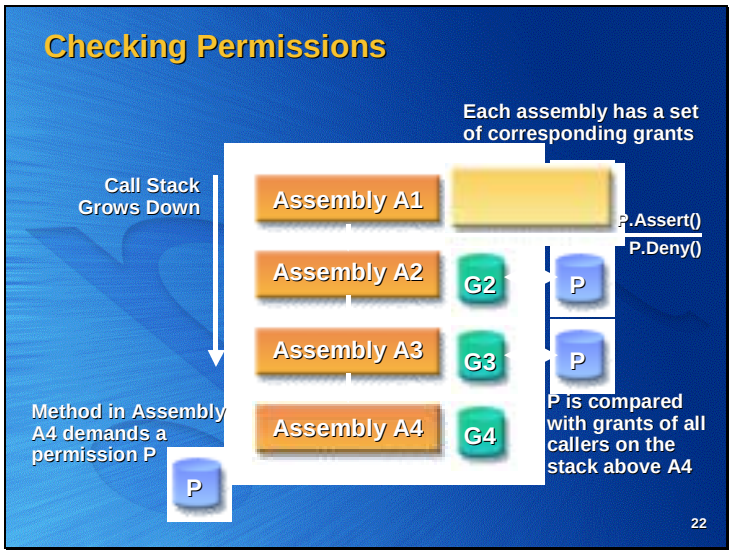
- Hosts specify “implicitly trusted” evidence.
- Custom membership conditions can interface with other authorization systems.

Example: ASP+/ISP application hosting

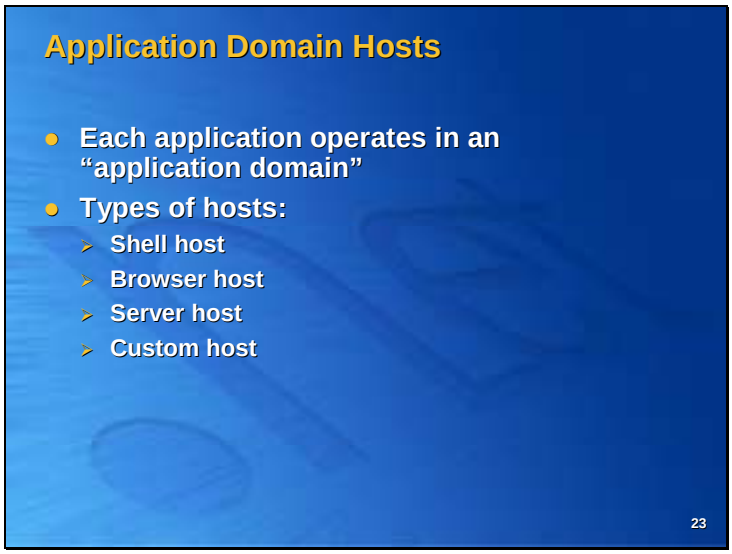
Application domains can limit policy for other domains they create.

Example: SQL Server™ user-defined assemblies

 ©2002 Microsoft Corporation



Checking Permissions



Application Domain Hosts

Each .NET application runs in an application domain under the control of a *host* that sets up the application domain and loads assemblies into it. The host has access to information about the code such as the zone from which the code originates or the digital signatures of assemblies in the application domain. A *trusted host* has permission to provide the runtime with evidence.

The following entities can be application domain hosts:

- **Shell host** – Launches applications (EXE's) from the shell.
- **Browser host** (Internet Explorer) – Operates code within the context of a web site.
- **Server host** (ASP .NET) – Operates code that handles requests submitted to a server.
- **Custom-designed hosts** – Creates domains and/or loads assemblies into domains including dynamic assemblies.

After creating a new application domain, a host can specify the policy it wants to be applied to code within the application domain. This policy is influenced by machine policy and user policy. A host can reduce the set of permissions allowed by machine and user policy, however, the application domain policy cannot expand the set of permissions allowed by machine and user policy. Policy can be set only once for an application domain and once application domain policy is set, further attempts to set the application domain policy will fail. To provide application domain policy, the host must be granted the SecurityPermission for controlling domain policy.

After application domain policy is set, all subsequently loaded assemblies will be granted permissions under the new policy (machine policy, user policy and application domain policy). Previously loaded assemblies will get grants under the pre-existing policy (i.e., machine and user policy only). Assemblies loaded into the application domain before the policy is in place will not have permission grants re-evaluated under the newly set application domain policy.

If the host is trusted, it can provide evidence to the runtime about assemblies loaded into the application domain. If a domain host does not have sufficient permission (i.e., the SecurityPermission for controlling evidence), the security enforced on the host is used to determine the security enforced on the assembly. In this situation, any evidence passed to CreateDomain() is ignored and the domain will share the same evidence as the host.

 ©2002 Microsoft Corporation

Typically, after the application domain is created, the host loads the first (main) assembly into the application domain and calls into it to begin execution. When a code in the first assembly references a code in another assembly, the loader resolves the reference, loads the appropriate assembly into the application domain and supplies the evidence about the assembly to the runtime.

The System.AppDomain class provides the application domain functionality used by hosts.

Miscellaneous Topics

Cryptography Namespace - The **System.Security.Cryptography** namespace provides cryptographic services, including secure encoding and decoding of data, as well as many other operations, such as hashing, random number generation, and message authentication.

Obfuscation (confusion) techniques.

Type-safe code accesses only the memory locations it is authorized to access.

The .NET Framework SDK supplies command-line utilities to help manage security tasks.

 ©2002 Microsoft Corporation

Cryptography Namespace

The cryptography namespace provides cryptographic services including secure encoding and decoding of data as well as many other operations, such as hashing, random number generation, message authentication and formation of digital signatures.

There are two main types of cryptographic algorithms: symmetric and asymmetric. Symmetric cryptographic algorithms (see SymmetricAlgorithm) have a single secret key that is used for both encryption and decryption. Use of a symmetric algorithm requires that only the sender and the receiver know the secret key. DES_CSP, RC2_CSP and TripleDES_CSP are implementations of symmetric algorithms.

Asymmetric Cryptographic Algorithms (see AsymmetricAlgorithm), also known as public-key algorithms, require that each entity maintains a private key and a public key. Both keys are unique to the entity. The public key can be made available to anyone and is used for encoding data to be sent to a receiver. The private key must be kept private and is used for decoding messages encoded using the receiver's public key. RSA_CSP is an implementation of a public-key algorithm.

Many public-key systems also can be used to form digital signatures to authenticate the identity of another entity and to protect the integrity of data. For example, to use a public-key system to digitally sign a message, the sender first applies a hash function to the message to create a digest. The sender then encrypts the message digest with the sender's private key to create the sender's personal signature. Upon receiving the message and signature, the receiver decrypts the signature by using the sender's public key to recover the message digest and hashes the message by using the sender's hash algorithm. If the message digest that the receiver computes matches the message digest received from the sender, the receiver can be sure the message came from the sender. Note that a signature can be verified by anyone because the sender's public key is common knowledge and this method does not retain the secrecy of the message. For the message to be secret, it must be encrypted as previously described. The receiver decrypts the message by using the receiver's private key and then computes the message digest as before. DSA_CSP is an implementation of a digital signature algorithm. **RSA_CSP** also can be used for creating and verifying a digital signature.

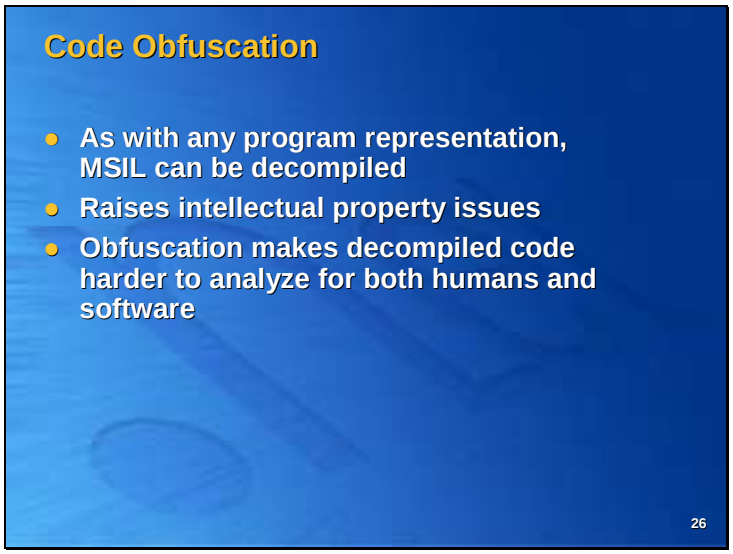
Hash functions (see HashAlgorithm) are fundamental to modern cryptography. These functions map binary strings of an arbitrary length to small binary strings of a fixed length, known as hash-values. A cryptographic hash function is computationally infeasible to find two distinct inputs that hash to the same

value. Hash functions are commonly used with digital signatures and for data integrity. MD5_CSP and SHA1Managed are implementations of hash algorithms.

Message Authentication Codes (MAC) provides data origin authentication and data integrity (see KeyedHashAlgorithm). HMACSHA1 and MAC_TripleDES are implementations of MAC algorithms.

Random number generation (see RandomNumberGenerator) is integral to many cryptographic operations. For example, cryptographic keys for encryption and decryption need to be as random as possible, so it is infeasible to reproduce them. Cryptographic random number generators must have the property that it is computationally infeasible to predict the next output bit from the generator with better than probability 0.5; that is, any method of predicting the next output bit must not perform better than random guessing. RNG_CSP is an implementation of a random number generator algorithm.

 ©2002 Microsoft Corporation


Code Obfuscation
As with any program representation, MSIL can be decompiled
Raises intellectual property issues
Obfuscation makes decompiled code harder to analyze for both humans and software
26

Code Obfuscation

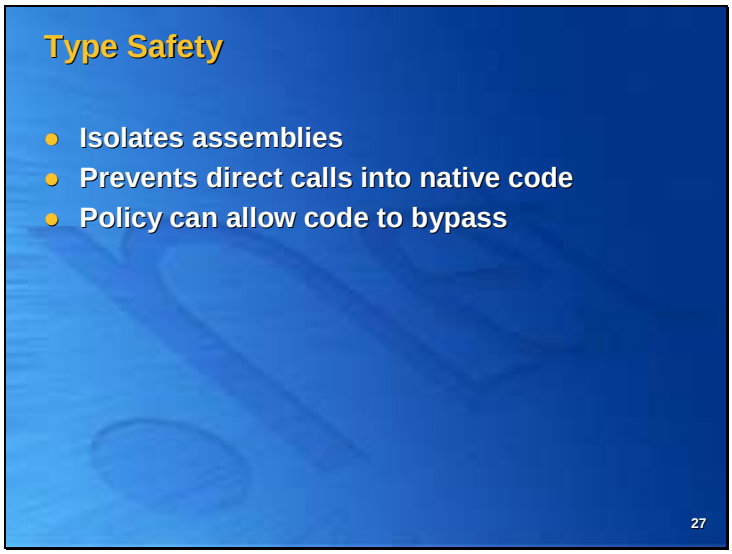
Having rich metadata and a high-level intermediate language is key to the .NET platform since it makes information available any time, any place on any device. This is what facilitates deployment- and run-time enforcement of type safety and true cross-language interoperability. Moreover, it forms the basis of the .NET Security mechanism.

As with any program representation, people can disassemble your code and Microsoft Intermediate Language (MSIL) is no exception. This naturally raises concerns about IP protection.

We often are asked if pre-jitting is an option here. No: Pre-jitting is NOT provided "as a way to protect your source." Rather, it is provided to provide a start-up performance improvement in certain scenarios. Right now, the runtime still requires that the original IL file be shipped (metadata is needed at load time; pre-jitting bakes in a number of assumptions that could fail and require re-jitting). So, while one could imagine that shipping pre-jitted code would, as a side-effect, address the IP issue to some degree by, essentially, shipping x86 code (that's what pre-jitted code is), the current requirement for the original IL file makes that not an option.

We are also asked about encryption. There are no current plans to support encryption of MSIL. In fact, even if you did encrypt it, we would need to decrypt it before execution (or change our engine) so we can load the classes and jit the code, so that alone won't keep the source from prying eyes completely.

Given what is acceptable on the market today in other environments, a very reasonable short-term solution is simple obfuscation of non-public names. It won't stop someone dead set on decompiling your code, but it does guard against giving away IP to casual inspection. Microsoft is likely to provide a basic obfuscator and hopes that the industry will expand and improve on the basic feature set with their own offerings.



Type Safety

Type-safe code accesses only the memory locations it is authorized to access. It cannot read, for example, values from another object's private fields. Type-safe code accesses types only in well-defined, allowable ways. During Just-In-Time (JIT) compilation, an optional verification process examines the Microsoft Intermediate Language (MSIL) in an attempt to verify that the MSIL is type-safe. This process is skipped if the code has permission to bypass verification.

Although verification of type safety is not mandatory, type safety plays a crucial role in assembly isolation and security enforcement. When code is type-safe, the runtime can completely isolate assemblies from each other; this isolation helps ensure that assemblies cannot adversely affect each other, and it increases application reliability. Type-safe components can execute safely in the same process even if they are trusted at different levels. When code is not type-safe, the runtime cannot prevent it from calling into native code to perform malicious operations. If code is type-safe, however, the runtime's security enforcement mechanism can ensure that it does not access native code unless it has permission.

 ©2002 Microsoft Corporation

Security Tools
The graphical tool for managing security policy is MSCORCFG. It is an MMC snap-in
CASPOL, cert2spc, Certmgr, Chktrust, permview, peverify, secutil, signcode
28

Security Tools

The .NET Framework SDK supplies command-line utilities that help you perform security-related tasks and test your components and applications before you deploy them. The following table briefly describes each of these tools:

Tool Name - Description

Assemble generation utility takes files containing MSIL and creates a PE (portable executable) file that contains an assembly manifest.

Caspol enables you to view and configure security policy. You can see what permissions are granted to a specified assembly and to which code groups that assembly belongs.

cert2spc creates a Software Publisher's Certificate (SPC) from one or more X.509 certificates.

Certmgr manages certificates, certificate trust lists (CTLs) and certificate revocation lists (CRLs).

Chktrust checks the validity of a signed file.

Makecert generates a X.509 certificate that can be used for testing purposes only.

Permview allows you to view an assembly's requested permissions.

Peverify determines whether the JIT compilation process can verify the type-safety of the assembly.

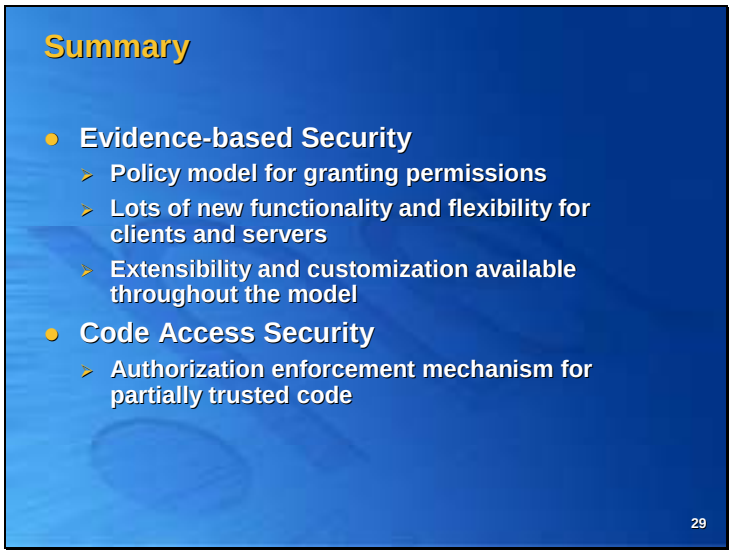
Secutil obtains the X.509 certificate and/or strong (shared) assembly names in a form that is easy to place in source files.

Setreg changes the registry settings that pertain to certificates and digital signatures.

Signcode enables you to digitally sign an assembly.

SN helps with key management and signature verification for strong (shared) assembly names.

Storeadm manages isolated storage, providing the ability to list the user's stores and delete them.


Summary
Evidence-based Security
Policy model for granting permissions
Lots of new functionality and flexibility for clients and servers
Extensibility and customization available throughout the model
Code Access Security
Authorization enforcement mechanism for partially trusted code
29

Summary

Security at the right level – Trusting code based on policy, the user and the code itself, not just who is operating it.

- Protection of intellectual property
- Provides verifiably safe code
- Code safety through Type Safety
- Provide tools to help perform security related tasks

 ©2002 Microsoft Corporation

Resources

How ASP Security Works
An overview of ASP Security
http://msdn.microsoft.com/library/dotnet/cpguide/cpconhowaspnetsecurityworks.htm

Key Concepts in Application Security
A basic review of the major components needed to secure applications
http://msdn.microsoft.com/library/dotnet/cpguide/cpconkeyconceptsinsecurity.htm

 ©2002 Microsoft Corporation